This listing of claims will replace all prior versions, and listings, of claims in the application.

Listing of Claims:

1. (original) A method for providing versioning support for at least one software component of an object-oriented programming language, the method comprising:

specifying programmer intent with regard to versioning of said at least one software component by assigning at least one keyword to said at least one software component.

2. (original) A method according to claim 1, wherein said assigning said at least one keyword includes assigning at least one of virtual, new and override keywords.

3. (original) A method according to claim 1, wherein said assigning said at least one keyword to said at least one software component specifies programmer intent with regard to whether said at least one software component overrides another software component.

4. (original) A method according to claim 1, wherein said assigning said at least one keyword to said at least one software component specifies programmer intent with regard to whether said at least one software component is capable of being overridden by another software component.

5. (original) A method according to claim 1, wherein said assigning said at least one keyword to said at least one software component specifies programmer intent with regard to whether said at least one software component hides another software component.

6. (currently amended) A method according to claim 1, wherein said at least one software component is at least one member of the object-oriented programming language and the object-oriented programming language is <u>from</u> one of <u>the sources of origin identified by</u> C#, Fortran, Pascal, Visual Basic, C, C++ and Java.

7. (original) A method according to claim 1, wherein said specifying of programmer intent includes assigning intelligent defaults to said at least one software component in the absence of assigning said at least one keyword to said at least one software component.

8. (original) A method according to claim 7, wherein when programmer intent is not fully specified, the compiler of the programming language produces a warning before assigning said intelligent defaults.

9. (original) A method according to claim 7, wherein said assigning of intelligent defaults includes assigning to said at least one software component the most limited form of accessibility, based upon the type of said at least one software component.

10. (original) A method according to claim 7, wherein by default, when said at least one software component is at least one method declaration with no accessibility modifiers appearing in the corresponding class, the at least one method declaration is defaulted to be private to that class.

11. (original) A method according to claim 7, wherein by default, said at least one software component is non-virtual, rather than virtual.

12. (original) A method according to claim 1, wherein said specifying of programmer intent includes providing a versioning-aware overload resolution method to locate a second method invoked by a first method invocation.

13. (original) A method according to claim 12, wherein said versioning-aware overload resolution method includes:

determining the type indicated by the first method invocation,

checking up the inheritance chain until at least one applicable, accessible, non-override method declaration is found;

performing overload resolution on the set of applicable, accessible, non-override methods

declared for the type; and

selecting the second method based on the performance of said overload resolution.

14. (original) A method according to claim 12, wherein for a virtual method invocation, said versioning-aware overload resolution method includes determining the second method based on the run-time type of the instance of the first method invocation.

15. (original) A method according to claim 12, wherein for a non-virtual method invocation, said versioning-aware overload resolution method includes determining the second method based on the compile-time type of the instance of the first method invocation.

16. (original) A method according to claim 12, wherein said versioning-aware overload resolution method includes bounding names at run-time, and not bounding offsets at compile-time.

17. (original) A method according to claim 12, wherein the overload resolution method prevents a base software component from breaking the functionality of a derived software component when versioning the base software component and such breaking is not intended by the programmer.

18. (original) A method according to claim 1, wherein said at least one software component is binary compatible with code utilizing other versions of said at least one software component.

19. (original) A method according to claim 1, wherein said at least one software component is source compatible with code utilizing other versions of said at least one software component.

20. (original) A computer readable medium bearing computer executable instructions for carrying out the method of claim 1.

21. (original) A modulated data signal carrying computer executable instructions for performing the method of claim 1.

22. (original) A computing device comprising means for performing the method of claim 1.

23. (original) A computer readable medium having stored thereon a plurality of computer-executable modules written in an object-oriented programming language, the computer executable modules comprising:

a versioning mechanism enabling a programmer to specify intent with regard to versioning of at least one software component by assigning at least one keyword to said at least one software component.

24. (original) A computer readable medium according to claim 23, wherein said assigning said at least one keyword includes assigning at least one of virtual, new and override keywords.

25. (original) A computer readable medium according to claim 23, wherein said assigning said at least one keyword to said at least one software component specifies programmer intent with regard to whether said at least one software component overrides another software component.

26. (original) A computer readable medium according to claim 23, wherein said assigning said at least one keyword to said at least one software component specifies programmer intent with regard to whether said at least one software component is capable of being overridden by another software component.

27. (original) A computer readable medium according to claim 23, wherein said assigning said at least one keyword to said at least one software component specifies programmer intent with regard to whether said at least one software component hides another software component.

28. (currently amended) A computer readable medium according to claim 23, wherein said at least one software component is at least one member of the object-oriented programming language and the object-oriented programming language is from one of the sources of origin identified by C#, Fortran, Pascal, Visual Basic, C, C++ and Java.

29. (original) A computer readable medium according to claim 23, wherein said specifying of programmer intent includes assigning intelligent defaults to said at least one software component in the absence of assigning said at least one keyword to said at least one software component.

30. (original) A computer readable medium according to claim 29, wherein when programmer intent is not fully specified, the compiler of the programming language produces a warning before assigning said intelligent defaults.

31. (original) A computer readable medium according to claim 29, wherein said assigning of intelligent defaults includes assigning to said at least one software component the most limited form of accessibility, based upon the type of said at least one software component.

32. (original) A computer readable medium according to claim 29, wherein by default, when said at least one software component is at least one method declaration with no accessibility modifiers appearing in the corresponding class, the at least one method declaration is defaulted to be private to that class.

33. (original) A computer readable medium according to claim 29, wherein by default, said at least one software component is non-virtual, rather than virtual.

34. (original) A computer readable medium according to claim 23, wherein said specifying of programmer intent includes providing a versioning-aware overload resolution method to locate a second method invoked by a first method invocation.

35. (original) A computer readable medium according to claim 34, wherein said versioning-aware overload resolution method includes:

determining the type indicated by the first method invocation,

checking up the inheritance chain until at least one applicable, accessible, non-override method declaration is found;

performing overload resolution on the set of applicable, accessible, non-override methods declared for the type; and

selecting the second method based on the performance of said overload resolution.

36. (original) A computer readable medium according to claim 34, wherein for a virtual method invocation, said versioning-aware overload resolution method includes determining the second method based on the run-time type of the instance of the first method invocation.

37. (original) A computer readable medium according to claim 34, wherein for a non-virtual method invocation, said versioning-aware overload resolution method includes determining the second method based on the compile-time type of the instance of the first method invocation.

38. (original) A computer readable medium according to claim 34, wherein said versioning-aware overload resolution method includes bounding names at run-time, and not bounding offsets at compile-time.

39. (original) A computer readable medium according to claim 34, wherein the overload resolution method prevents a base software component from breaking the functionality of a derived software component when versioning the base software component and such breaking is not intended by the programmer.

40. (original) A computer readable medium according to claim 23, wherein said at least one software component is binary compatible with code utilizing other versions of said at least one software component.

41. (original) A computer readable medium according to claim 23, wherein said at least one software component is source compatible with code utilizing other versions of said at least one software component.

42. (original) An object-oriented programming language for producing computer executable modules, comprising:

a versioning mechanism enabling a programmer to specify intent with regard to versioning of at least one software component by assigning at least one keyword to said at least one software component.

43. (original) An object-oriented programming language according to claim 42, wherein said assigning said at least one keyword includes assigning at least one of virtual, new and override keywords.

44. (original) An object-oriented programming language according to claim 42, wherein said assigning said at least one keyword to said at least one software component specifies programmer intent with regard to whether said at least one software component overrides another software component.

45. (original) An object-oriented programming language according to claim 42, wherein said assigning said at least one keyword to said at least one software component specifies programmer intent with regard to whether said at least one software component is capable of being overridden by another software component.

46. (original) An object-oriented programming language according to claim 42, wherein said assigning said at least one keyword to said at least one software component specifies programmer intent with regard to whether said at least one software component hides another software component.

47. (currently amended) An object-oriented programming language according to claim 42, wherein said at least one software component is at least one member of the object-oriented programming language and the object-oriented programming language is from one of the sources of origin identified by C#, Fortran, Pascal, Visual Basic, C, C++ and Java.

48. (original) An object-oriented programming language according to claim 42, wherein said specifying of programmer intent includes assigning intelligent defaults to said at least one software component in the absence of assigning said at least one keyword to said at least one software component.

49. (original) An object-oriented programming language according to claim 48, wherein when programmer intent is not fully specified, the compiler of the programming language produces a warning before assigning said intelligent defaults.

50. (original) An object-oriented programming language according to claim 48, wherein said assigning of intelligent defaults includes assigning to said at least one software component the most limited form of accessibility, based upon the type of said at least one software component.

51. (original) An object-oriented programming language according to claim 48, wherein by default, when said at least one software component is at least one method declaration with no accessibility modifiers appearing in the corresponding class, the at least one method declaration is defaulted to be private to that class.

52. (original) An object-oriented programming language according to claim 48, wherein by default, said at least one software component is non-virtual, rather than virtual.

53. (original) An object-oriented programming language according to claim 42, wherein said specifying of programmer intent includes providing a versioning-aware overload resolution method to locate a second method invoked by a first method invocation.

54. (original) An object-oriented programming language according to claim 53, wherein said versioning-aware overload resolution method includes:

determining the type indicated by the first method invocation,

checking up the inheritance chain until at least one applicable, accessible, non-override method declaration is found;

performing overload resolution on the set of applicable, accessible, non-override methods declared for the type; and

selecting the second method based on the performance of said overload resolution.

55. (original) An object-oriented programming language according to claim 53, wherein for a virtual method invocation, said versioning-aware overload resolution method includes determining the second method based on the run-time type of the instance of the first method invocation.

56. (original) An object-oriented programming language according to claim 53, wherein for a non-virtual method invocation, said versioning-aware overload resolution method includes determining the second method based on the compile-time type of the instance of the first method invocation.

57. (original) An object-oriented programming language according to claim 53, wherein said versioning-aware overload resolution method includes bounding names at run-time, and not bounding offsets at compile-time.

58. (original) An object-oriented programming language according to claim 53, wherein the overload resolution method prevents a base software component from breaking the functionality of a derived software component when versioning the base software component and such breaking is not intended by the programmer.

59. (original) An object-oriented programming language according to claim 42, wherein said at least one software component is binary compatible with code utilizing other versions of said at least one software component.

60. (original) An object-oriented programming language according to claim 42, wherein said at least one software component is source compatible with code utilizing other versions of said at least one software component.